In the Specification:

Please amend the Specification as shown below. The paragraph numbers below correspond to the Application *as published* (Pub. No. US 2003/0046395 A1) since it appears most of the errors set forth below were the result of publishing errors. Applicant respectfully submits that the proposed amendments to the Specification are to correct various informalities, and that no new matter is being added.

Please replace Paragraph **[0002]** with the new paragraph shown below:

[0002] A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document [[orthe]] or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Please replace Paragraph **[0003]** with the new paragraph shown below:

[0003] The invention relates ~~generallyto~~ generally to event notification mechanisms for use in transactional servers.

Please replace Paragraph **[0008]** with the new paragraph shown below:

[0008] The Tuxedo product referred to herein is described in detail in the "BEA TUXEDO Reference Manual", herein incorporated by reference. The Object Management Group (OMG) Notification Service specification is described in detail in "Notification Service: OMG Technical Committee Document telecom/98-06-15", herein incorporated by reference. The Common Object Request Broker (CORBA) architecture is described in detail in "The Common Object Request Broker: Architecture and Specification, Revision 2.2, February 1998", herein incorporated by reference. The CORBA Event Service specification is described in detail in "~~CORBAservice~~ CORBA service: chapter 4, Event Service Specification, March 1997", herein incorporated by

reference.

Please replace Paragraph **[0016]** with the new paragraph shown below:

[0016] If a subscriber signs up for best effort delivery, an event channel (e.g., the "company" sending out catalogs) sends the events to the subscriber using one way messages (that is, by "bulk mail"), because it's fast and doesn't consume many system resources. However, because of the way most traditional subscription based systems (including ~~CORBA)work~~ CORBA) work, the channel is never told if the subscriber has gone away, so the channel can't automatically get rid of the subscription. Instead, the channel ends up wasting a lot of time sending events to subscribers who no longer exist (such as the analogy in which mail catalogs are sent to customers who have moved or died).

Please replace Paragraph **[0030]** with the new paragraph shown below:

[0030] If a cache entry does ~~existforthis~~ exit for this subscription, then the callback object reference is readily available. When dealing with a Transient Subscription, each subsequent lookup for a certain subscription id determines whether the next call to deliver an event to the callback ought to be a two-way or one-way, based on the cache parameters. After each two-way call, subsequent calls to deliver the event to that subscriber will be one-way, until either the max time between two-way calls has elapsed or the max number of one-way calls has been reached. If this is a two-way call the oneway_count is set to 0 and the ~~last_twoway_ume~~ last_twoway_time set to the current time. If this is a one-way call, the oneway_count is incremented and last_twoway_time left unaltered. A lookup is made on the subscription cache for this subscription. If the lookup determines this ought to be a two-way call, push_structured_event is directly invoked to deliver the event. If an error occurs during this invoke, the subscription is dropped, and the corresponding cache_entry is removed. If this must be a one-way call however, then DII is use to invoke push_structured_event and deliver the event. Since a one-way call was used, the system cannot tell if the event was delivered or not. Therefore, there is no need to cleanup dead

subscriptions until the next two-way call. Having obtained the callback object corresponding to the subscription id, the event is delivered to the callback.

Please replace Paragraph **[0069]** with the new paragraph shown below:

[0069] Suppliers use a push paradigm 108. They call a method (named push) in the Event Broker. The Event Broker takes responsibility for filtering and delivering the event. Consumers may specify a Quality Of Service (QOS) which effects the persistence of the Consumers subscription and, in the case of push Consumers, whether or not events delivery is retried following a failed delivery. There is no direct association between Suppliers and Consumers. At any point in time there may be zero, one or many Suppliers ~~and/orConsumers~~ and/or Consumers. The Event Service provides a variety of interfaces (described in detail below) to allow a subscriber or poster to interact with the Event Server and to post or receive events.

Please replace Paragraph **[0085]** with the new paragraph shown below:

[0085] The Fixed Header section consists of three fields: domain_type, event_type and an [[event_name]] event_name.

Please replace Paragraph **[0104]** with the new paragraph shown below:

[0104] 1. ~~Theapplication~~ The application posts (21) a structured ~~eventtothe~~ event to the Event Server Event Service (either in a transaction or not) via the "nts" server 130.

Please replace Paragraph **[0111]** with the new paragraph shown below:

[0111] **Figure 7** shows a flowchart illustrating a more detailed view of one ~~implmentation~~ implementation of the best effort delivery process used by the invention, in which the Event Server uses an event delivery timer to track the status of the delivery on a periodic basis, and to take

action when delivery falls outside of the best-effort delivery variables or administrative limits. In step 202, the subscriber subscribes to events via a transient subscription. In step 204 the event delivery timer is initialized, with an initial value for event check = 1. The subscription is then maintained as each event is delivered (step 206) using a single message process. In step 208, as each event delivered, the event check timer is incremented. A system administrator can preset a maximum value for the event check timer that is appropriate for the QOS and optimization reasons. When the event check timer reaches this maximum value (step 210) it checks for the status of the transient subscription by sending a two-way event delivery (step 212). If the two-way message fails (step 214) , then the system assumes the subscription is no longer valid, and it is terminated (step 216) and cleaned-up, as described above.

Please replace Paragraph **[0152]** with the new paragraph shown below:

[0152] Persistent subscriber: the events eventually end up in the error queue. Use Event Server ~~eadmin~~ admin to detect and cleanup.